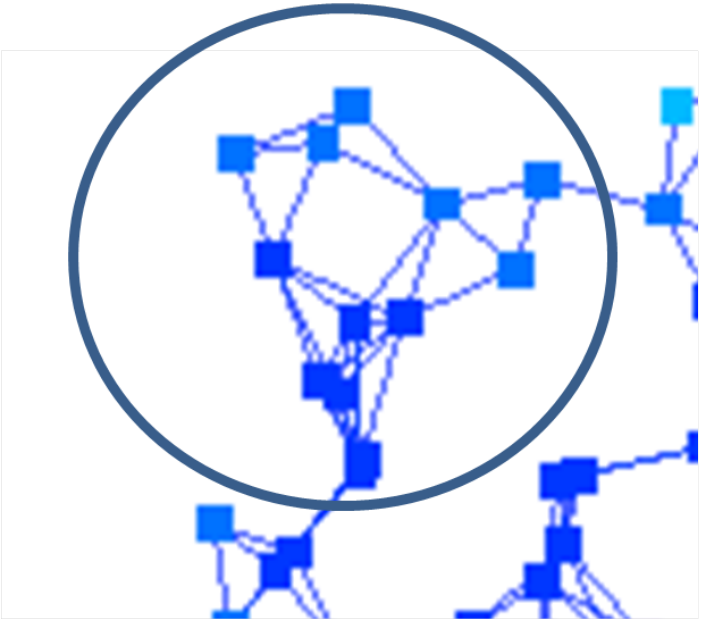
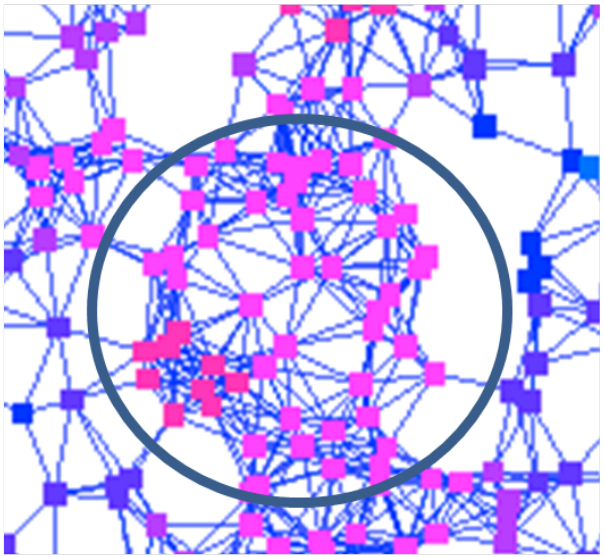
Finding Dense Subgraphs with Size Bounds

Reid Andersen Kumar Chellapilla
Microsoft Live Labs

Density of a Subgraph





Definition

The *density* of an induced subgraph $S \subseteq V$ is

$$d(S) = \frac{\texttt{edges}(S)}{|S|} = \frac{\text{number of edges in the induced subgraph}}{\text{number of nodes in the induced subgraph}}.$$

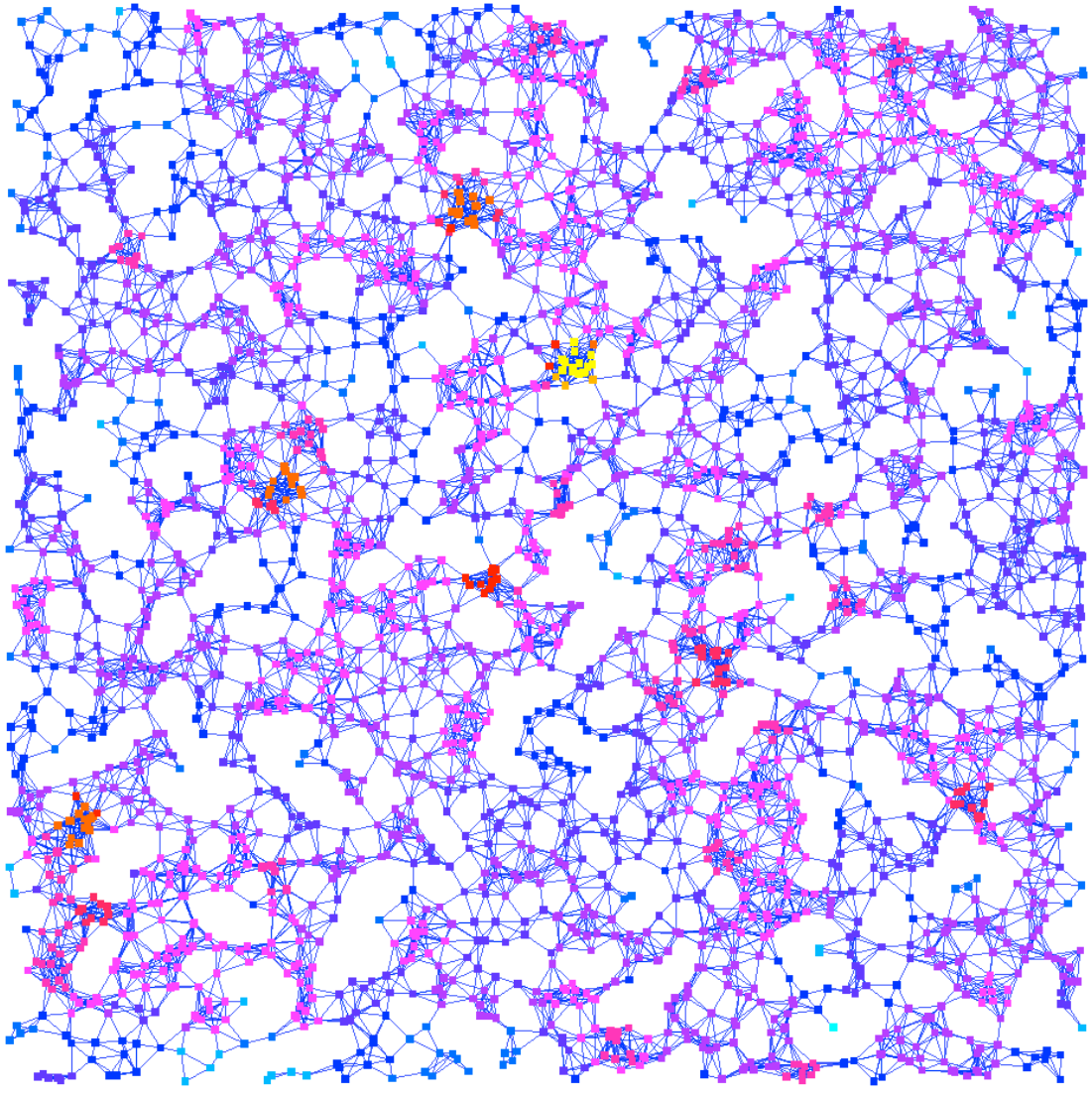
Example and Applications

Example of a dense subgraph: In an incidence graph between movies and actresses derived from the Internet Movie Database, the densest subgraph contains 2754 movies from 1935-1945: Easy Living (1937), This Is My Affair (1937), The Roaring Twenties (1939), Happy Go Lucky (1943), The Lodger (1944)...
On average, each actress in the subgraph appeared in 14 of these movies.

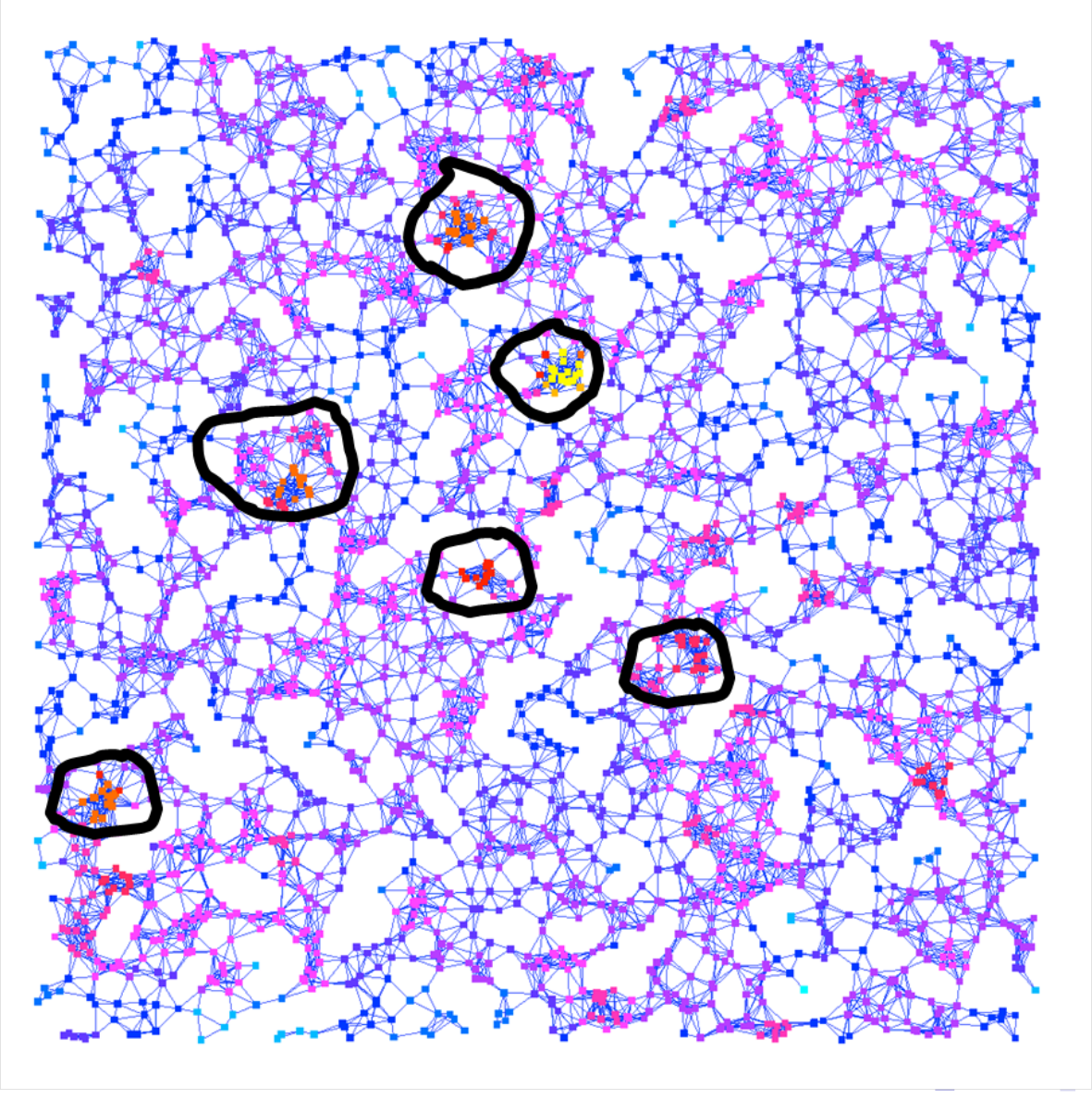
Previous work on finding dense subgraphs or near-cliques in web graphs:

- finding web communities [Dourisboure et al., WWW'07]
- finding link farms [Gibson et al., VLDB'04]
- finding bipartite cliques [Kumar et al., WWW'99]
- finding cliques for graph compression [Buehrer/Chellapilla, WSDM'08]

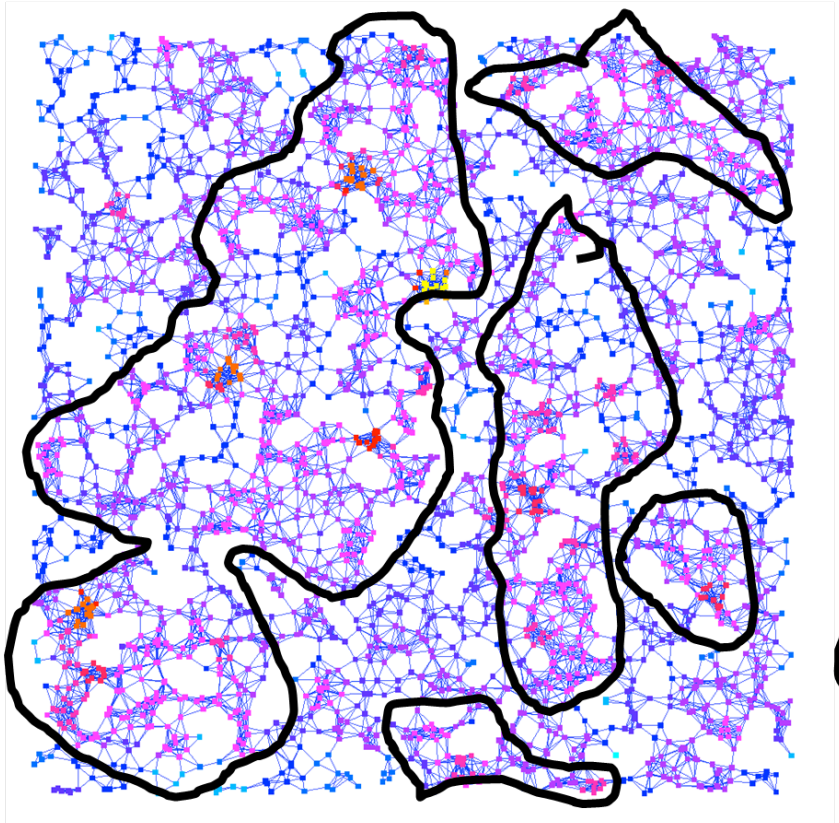
Example: dense regions in a geometric graph

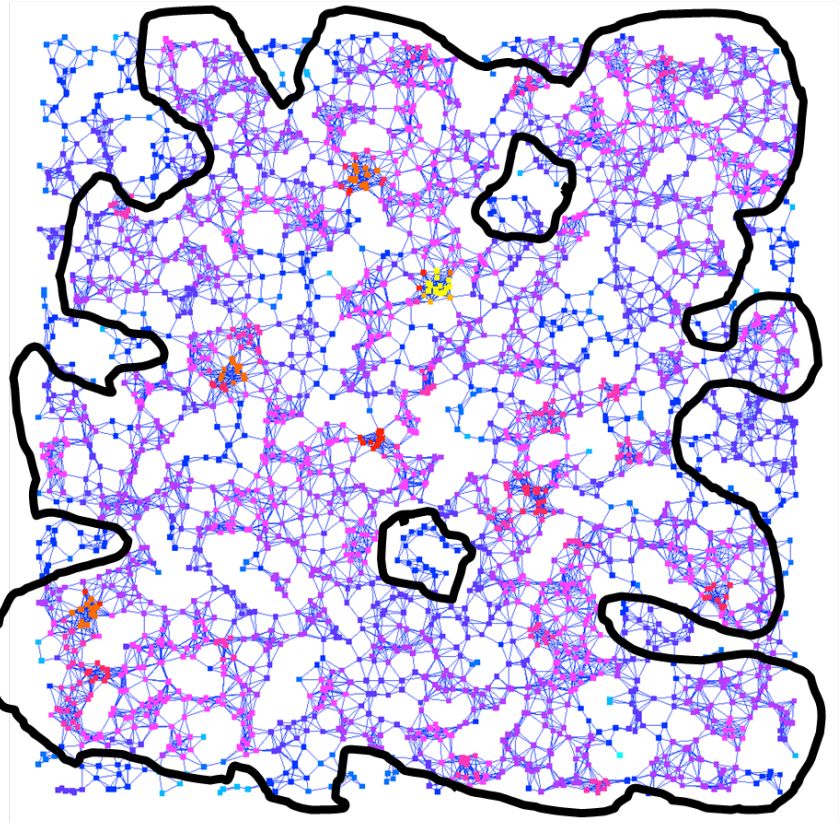


Finding small dense subgraphs near target vertices



Finding a large dense subgraph





Preprocess a graph by keeping only a large dense subgraph.

- Save time (e.g. computing PageRank on the 2-core)
- Restrict attention to the important parts (e.g. a high-value submarket in a sponsored search spending graph).

Two well-studied problems about dense subgraphs

densest subgraph

Find the densest subgraph in the input graph.

- Can be solved exactly in polytime using parametric flow. [Goldberg 84], [Gallo et al. 89].
- A set with 1/2 the optimal density can be found in linear time, using a greedy algorithm (the core decomposition). [Kortsarz/Peleg 92].

densest k-subgraph

Find the densest subgraph with exactly k vertices.

- NP-complete even for graphs with maximum degree 3. Best algorithm known has approximation ratio $n^{1/3-\delta}$. [Feige/Seltser 97], [Feige/Peleg/Kortsarz 01]
  Best hardness result says there's no PTAS [Khot]

Main question of this talk

We introduce two relaxations of the `densest` k`-subgraph` problem, and try to answer whether they are easy or hard.

`densest` k`-small-subgraph`
Find a subgraph on at most k vertices that has the highest density among all such subgraphs.

`densest` k`-large-subgraph`
Find a subgraph on at least k vertices that has the highest density among all such subgraphs.

Results

The `densest` k-`large-subgraph` can be approximated well.

- We give a 1/3-approximation algorithm: linear time greedy algorithm, based on the core decomposition [Seidman '83], extends the result of [Kortsarz, Peleg '92].
- We give a polynomial time 1/2-approximation algorithm based on parametric flow.
- Experimental results on publicly available web graphs.

The `densest` k-`small-subgraph` problem is almost as hard to approximate as the `densest` k-`subgraph` problem.

- NP-complete by reduction from `max-clique`. (easy)
- Given a polynomial time approximation algorithm for `densest` k-`large-subgraph` with ratio $1/\gamma$, we can construct a polynomial time approximation algorithm for `densest` k-`subgraph` with ratio $1/\gamma^2$.

How hard is it to find small dense subgraphs?

Definition

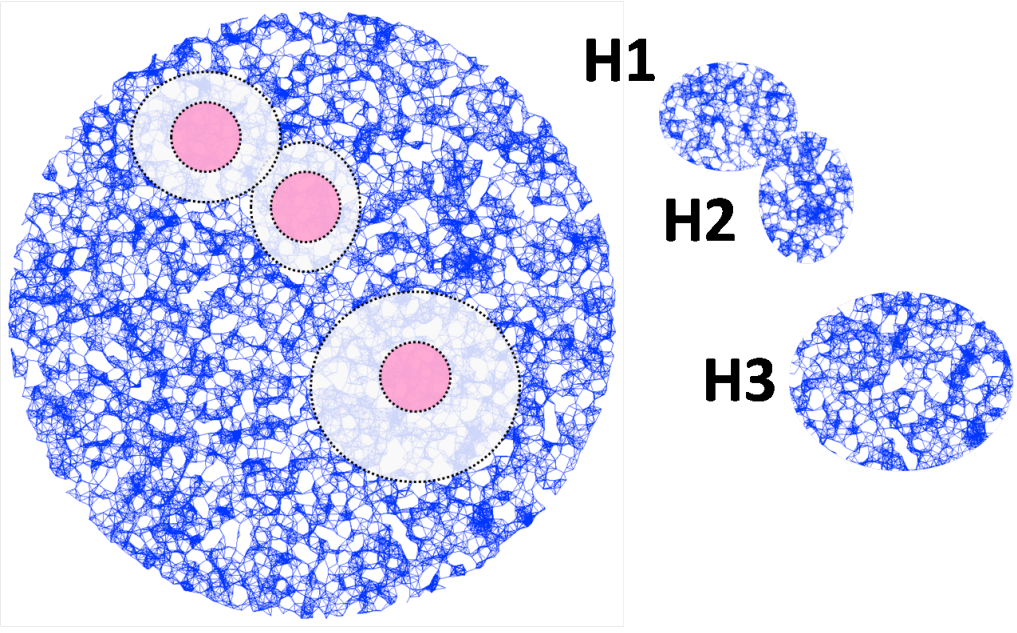
An algorithm is a (β, γ)-*algorithm* for the `densest` k`-small-subgraph` problem if it returns, for any input graph G and integer k, an induced subgraph of G with

- at most βk vertices. $(\beta \geq 1)$
- density at least γ times the optimal set on at most k vertices. $(\gamma \leq 1)$.

Theorem

If there is a polynomial time (β, γ)*-algorithm for* `densest` k`-small-subgraph` *problem, then there is a polynomial time approximation algorithm for the* `densest` k`-subgraph` *problem with ratio* $(\gamma \min(\gamma, \beta^{-1})/8)$.

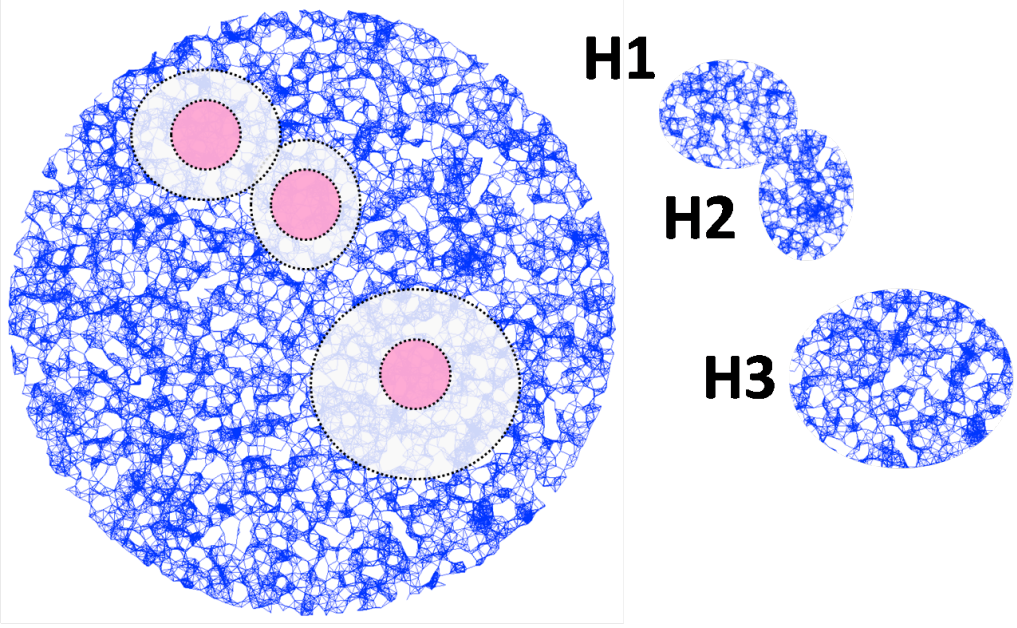
Proof idea



To find a dense subgraph on exactly k vertices:

- Find a dense subgraph on at most βk vertices using your algorithm for `densest` k`-small-subgraph`.
- Remove all the edges from that subgraph from the graph.
- Repeat, removing subgraphs $H_1, H_2, \ldots$ until you have removed all the edges.

Proof idea

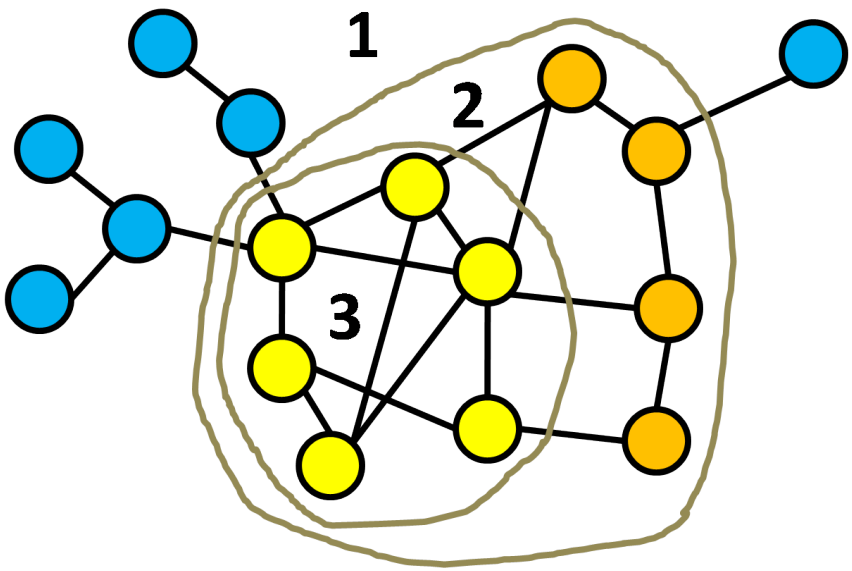


- Consider the first time when the number of edges you have removed is at least half the number of edges in the optimal subgraph with exactly k vertices.
- If that removed subgraph has $< k$ nodes, pad it with arbitrary vertices to make a set of size k. If the subgraph has $> k$ nodes, greedily remove the smallest degree vertex until you have a set of size k.

Finding large dense subgraphs using the core decomposition

Definition

$\texttt{core}(G, d)$ is the unique largest induced subgraph of G whose vertices all have degree at least d.



[Seidman '83] [Kortsarz/Peleg 92] [Charikar 00]

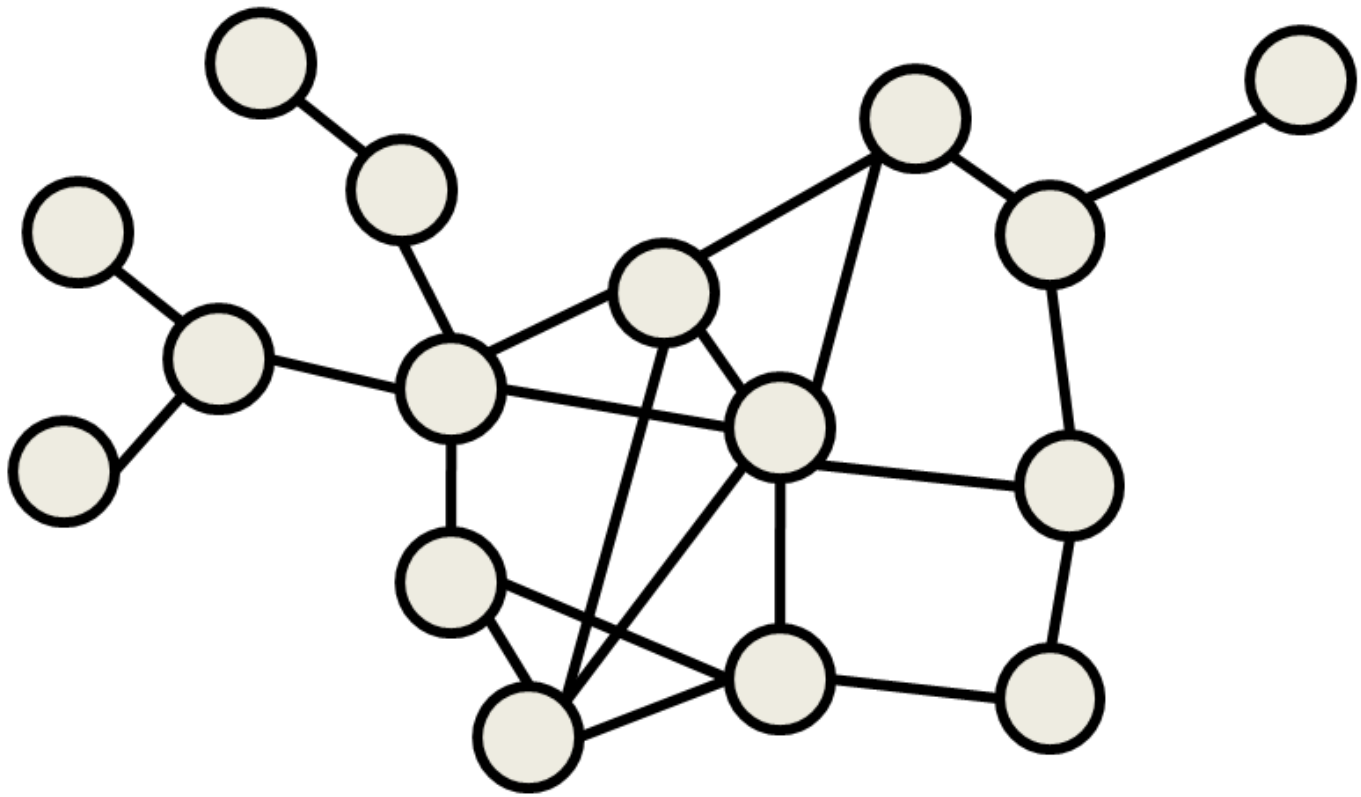
Core decomposition algorithm

`CoreOrdering`(G) :
Output: a list of vertices in the order $v_n \dots v_1$.

1. Let $G_n = G$. Repeat until $G_0 = \emptyset$:
2. Pick a vertex v_i that minimizes $\texttt{degree}(v_i, G_i)$.
3. Remove v_i and its edges from G_i to form G_{i-1}.
4. Charge v_i for the edges that get removed.
   $\texttt{charge}(v_i) = \texttt{degree}(v_i, G_i)$.

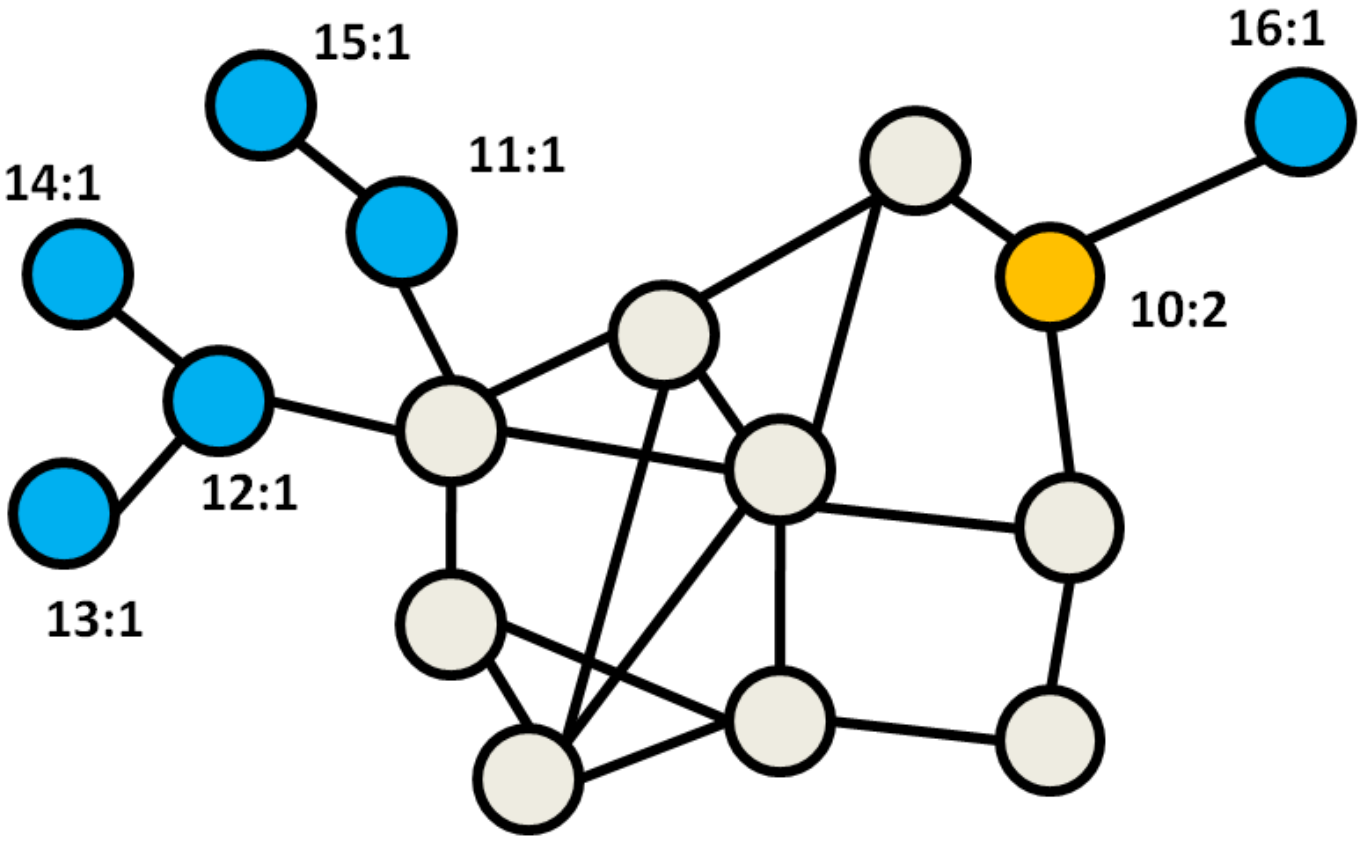
- Let $I(d)$ be the index of the first node that is charged at least d. Then $\texttt{core}(G, d) = \{v_1, \dots, v_{I(d)}\}$.
- The core ordering can be computed in time $O(m + n)$. Keep each vertex in a bucket corresponding to its current degree. When a node is removed, update its neighbors.

Core decomposition example



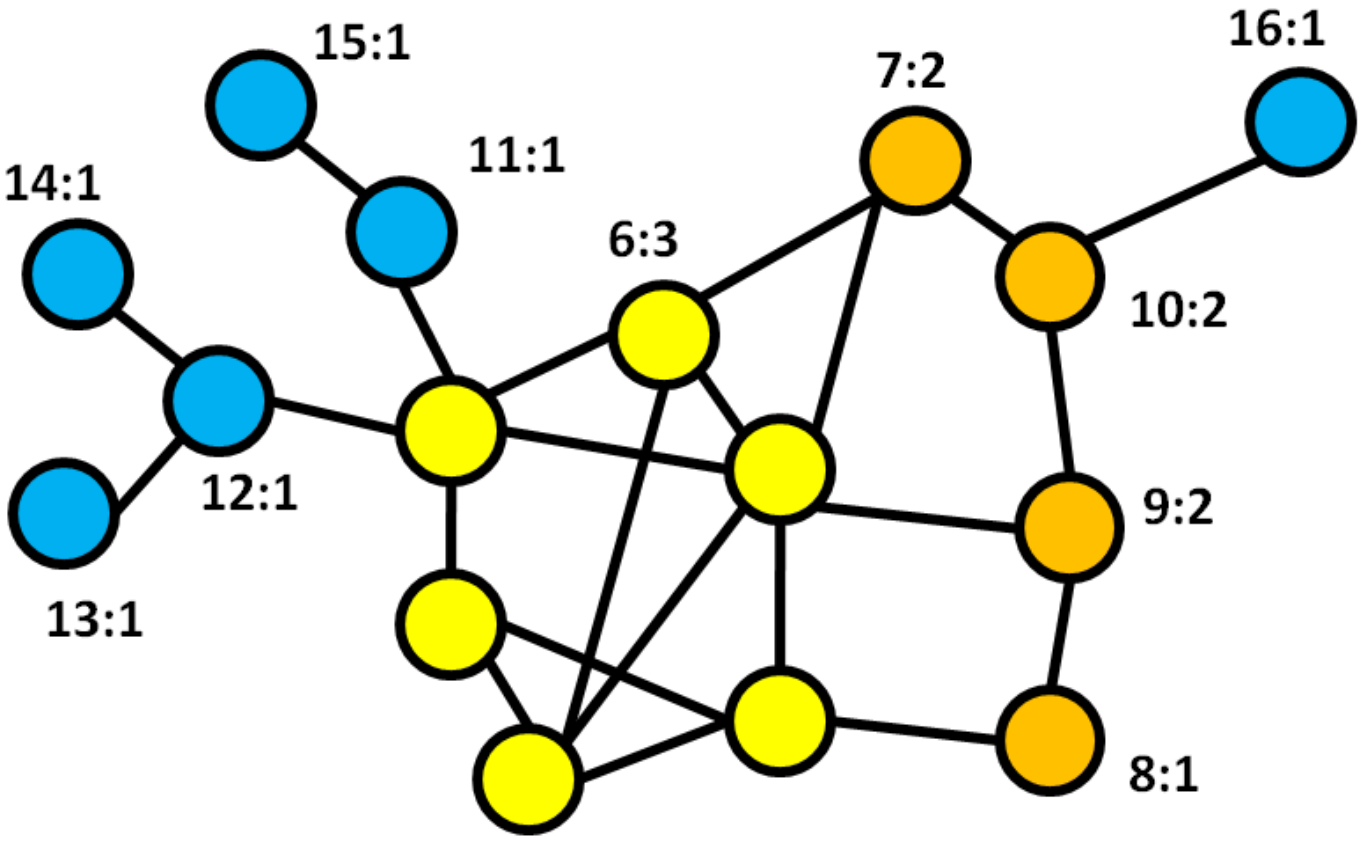
index															
charge															

Core decomposition example



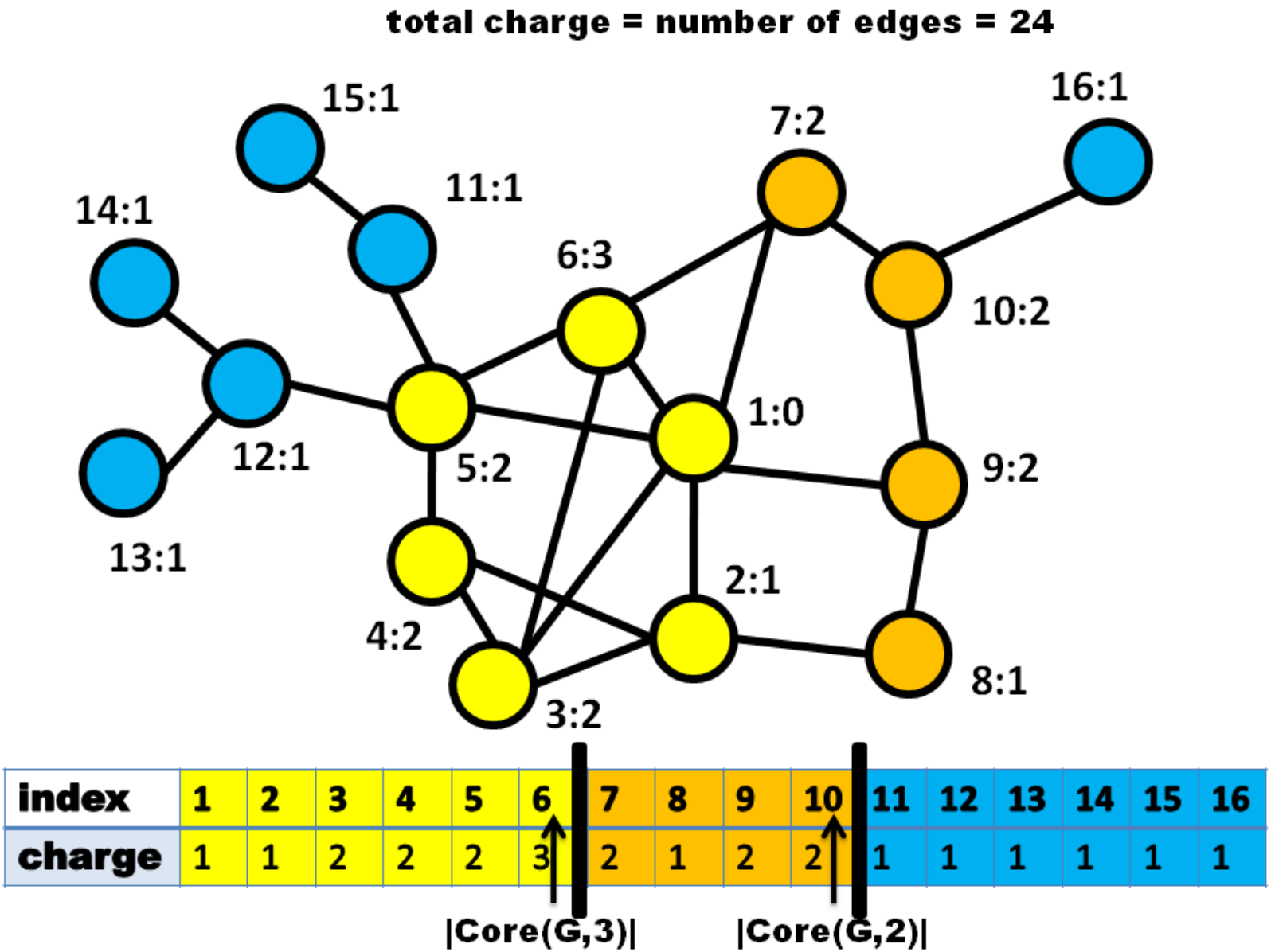
index									10	11	12	13	14	15	16
charge									2	1	1	1	1	1	1

Core decomposition example



index						6	7	8	9	10	11	12	13	14	15	16
charge						3	2	1	2	2	1	1	1	1	1	1

Core decomposition example



Algorithm for finding large dense subgraphs

`LargeDense`(G, k) :
Input: a graph G with n vertices, and an integer k.
Output: an induced subgraph of G with at least k vertices.

1. Compute the core ordering $v_1 \dots v_n$.
2. Compute the density of each subgraph $H_i = \{v_1 \dots v_i\}$.
3. Output the densest subgraph H_i for which $i \geq k$.

Theorem

`LargeDense`(G, k) *is a (1/3)-approximation algorithm for the* `densest` k`-large-subgraph` *problem.*

the running time of `LargeDense`(G, k) is $O(m + n)$.

Sketch of the proof

Lemma

For any graph H with density D, and any parameter $\alpha \in [0,1]$,

$$\texttt{edges}(\texttt{core}(H, \alpha D)) \geq (1-\alpha)\texttt{edges}(H).$$

Proof of Lemma. Let $J = |\texttt{core}(H, \alpha D)|$.

$$\begin{aligned}
\texttt{edges}(H) &= \texttt{charge}(v_n, \ldots, v_1) \\
&= \texttt{charge}(v_n, \ldots, v_k) + \texttt{charge}(v_{k-1}, \ldots, v_1) \\
&\leq n\alpha D + \texttt{edges}(\texttt{core}(H, \alpha D)).
\end{aligned}$$

Then, apply this lemma to the densest induced subgraph of G on at least k vertices, with $\alpha = 2/3$.

Experiments: graphs and running time

We tested `LargeDense` on three page-level web graphs: webbase-2001, uk-2005, cnr-2000, from the WebGraph framework provided by the Laboratory for Web Algorithmics. Also, one domain graph snapshot from Microsoft: domain-2006

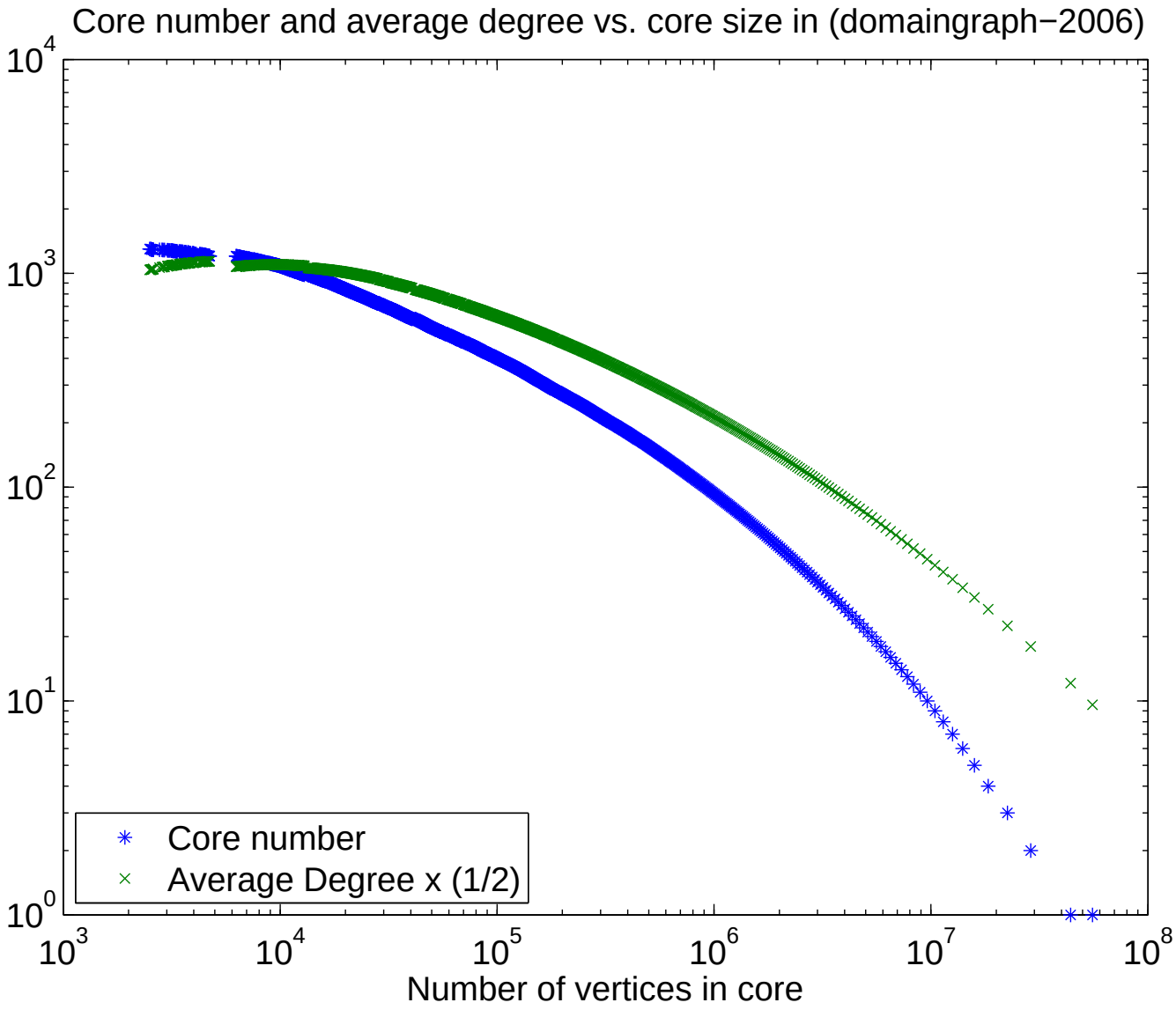
We treated each directed arc as an undirected edge.

The algorithm was implemented in C++/STL, and run on a commodity server.

graph	num nodes	total degree	run time (sec)
domain-2006	55,554,153	1,067,392,106	263.81
webbase-2006	118,142,156	1,985,689,782	204.573
uk-2005	39,459,926	1,842,690,156	92.271
cnr-2000	325,558	6,257,420	0.359

Figure: Graph size and time required to compute the core order

Size of core vs. core number and density (Domain graph)



Approximating the densest k-subgraph

- No good algorithms are known for finding the densest subgraph on exactly k vertices.
- But, the previous plot indicates that for one specific graph, the set $\{v_1 \dots v_k\}$ is a good approximation of the densest k-subgraph for all k above a certain small threshold:
  - For all $k \geq k_*$, get 1/3 of the optimal density on k vertices.
  - For all $k \geq k_{**}$, get 1/4 of the optimal density on k vertices.

graph	num nodes (n)	k_*	k_{**}
domain-2006	55,554,153	9,445	2,502
webbase-2001	118,142,156	48,190	1,219
uk-2005	39,459,926	368,741	587
cnr-2000	325,558	13,237	82

Figure: Comparison of k_* and n

When do we get a good approximation of the densest k-subgraph?

We introduce a graph parameter k_*. Intuitively, k_* describes how small a core of the graph must be before it can be nearly degree-regular.

Definition

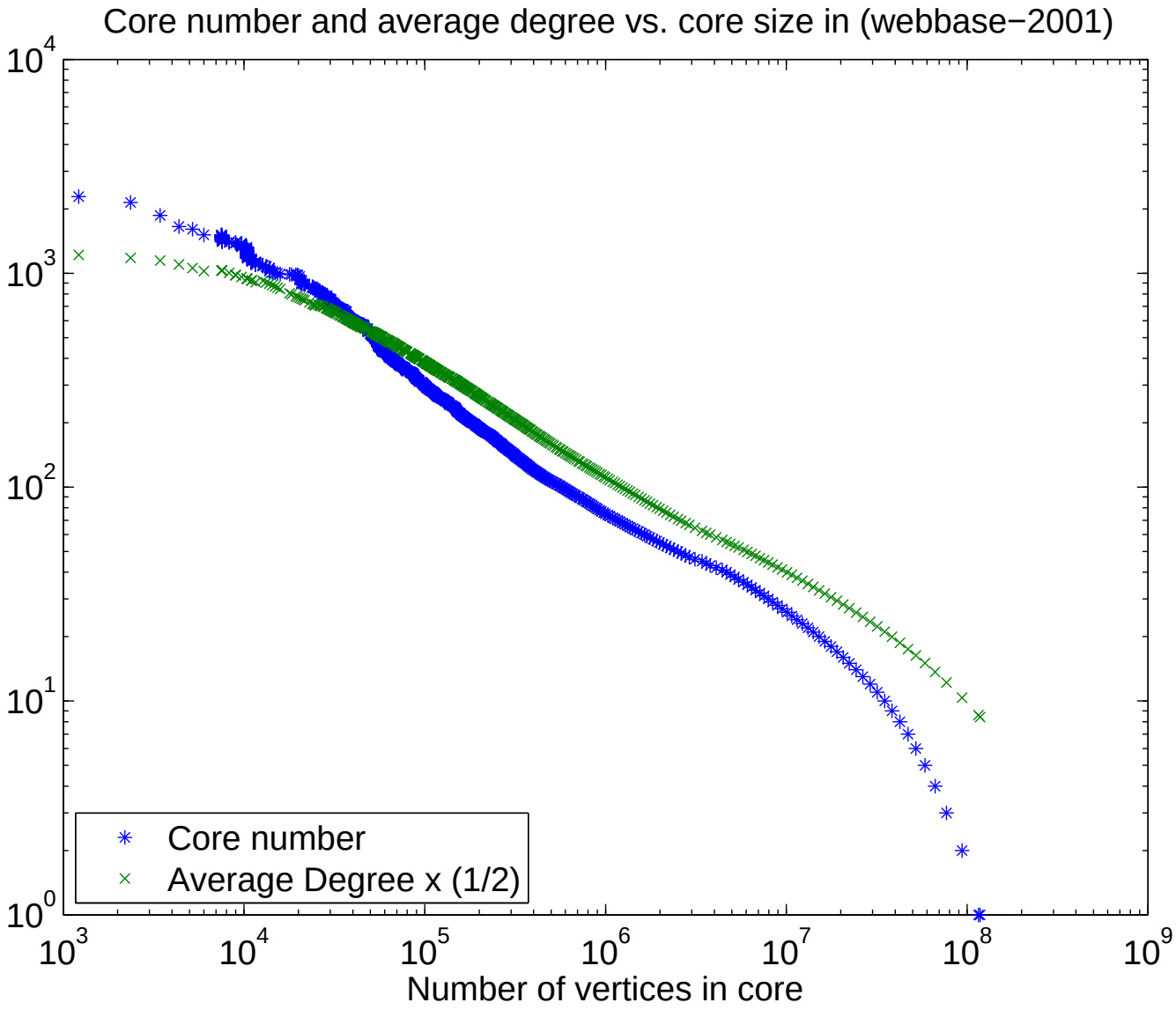
For a given graph G,

- Let d_* be the smallest value such that the average degree of the core $\texttt{core}(d_*)$ is less than $2d_*$.
- Let $k_*(G) = |\texttt{core}(d_*)|$ be the number of vertices in that core.

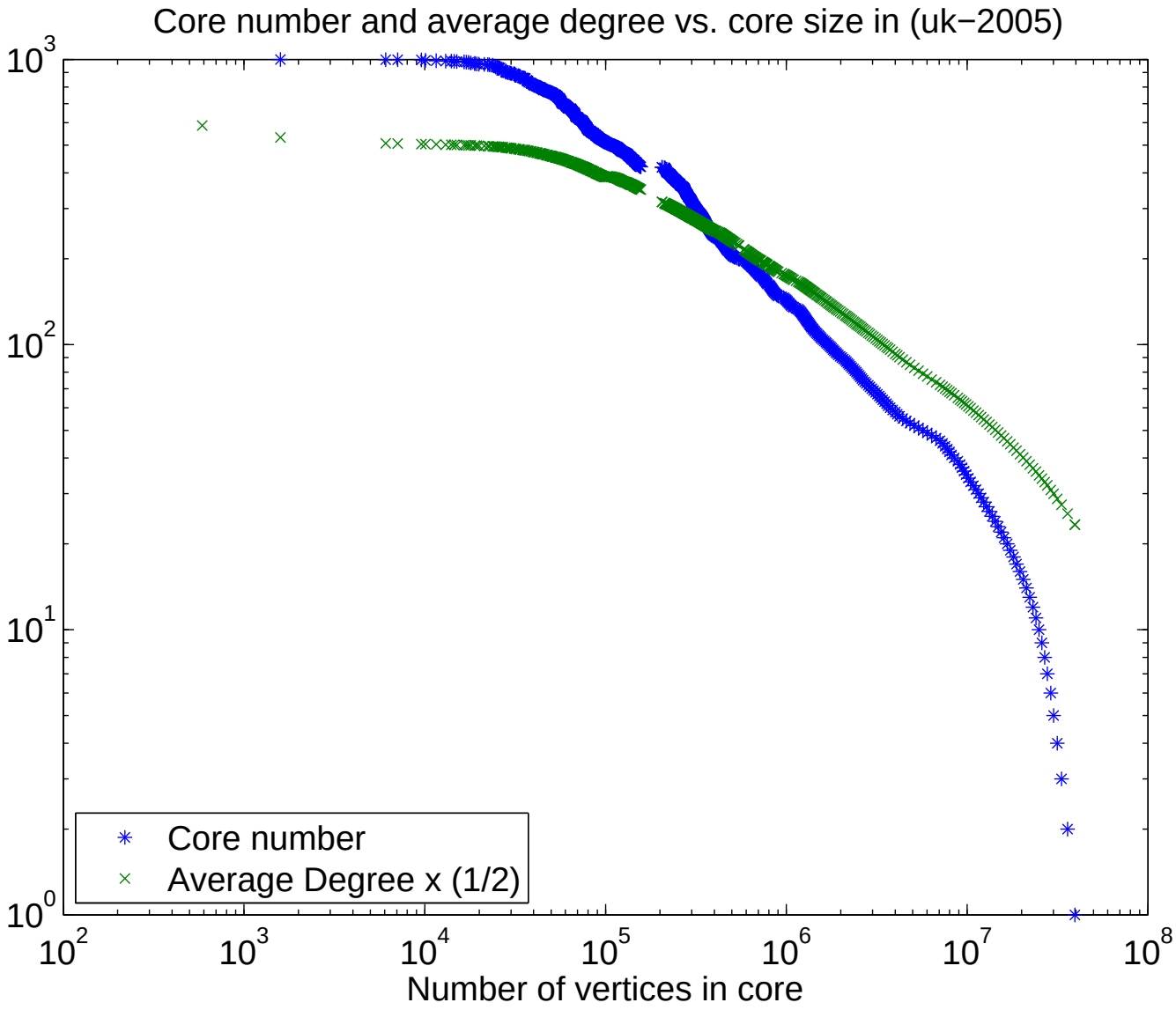
Theorem

For all $k \geq k_*$, the top k nodes in the core ordering have at least $1/3$ the density of the densest subgraph on k vertices.

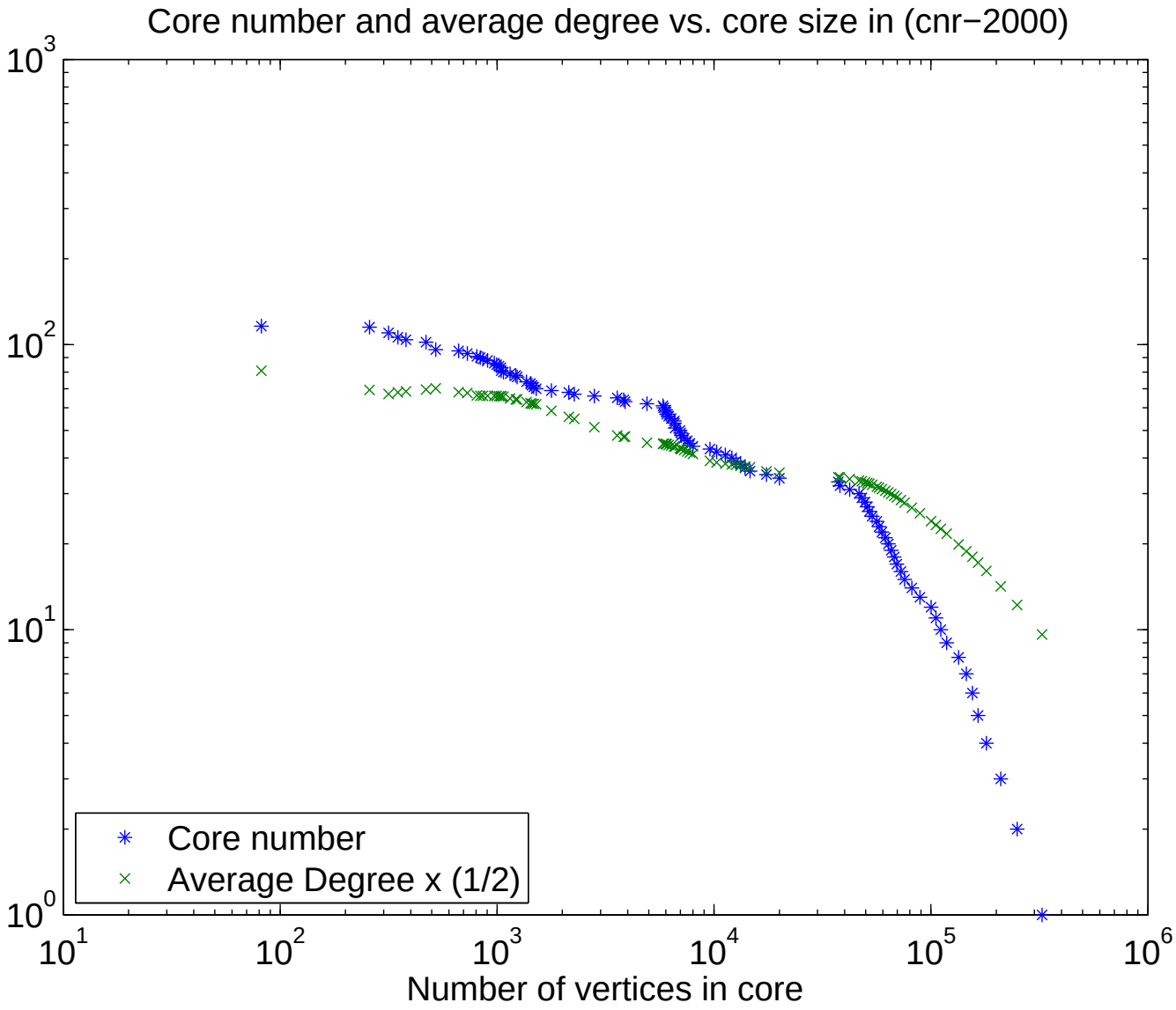
Size of core vs. core number and density (graph: webbase 2001)



Size of core vs. core number and density (graph: uk2005)



Size of core vs. core number and density (graph: cnr2000)



Noteworthy cores

graph	core number	nodes in core	density
domain-2006			
k_* core	1,099	9,445	2196.32
densest core	1,203	4,737	2275.96
highest numbered core	1,298	2,502	2072.42
webbase-2001			
k_* core	548	48,190	1089.42
highest numbered core	2,281	1,219	2436
uk-2005			
k_* core	258	368,741	515.851
highest numbered core	1,002	587	1171.98
cnr-2000			
k_* core	38	13,237	75.1145
highest numbered core	116	82	161.976

Miscellaneous section

Core ordering in a sponsored search bidding graph

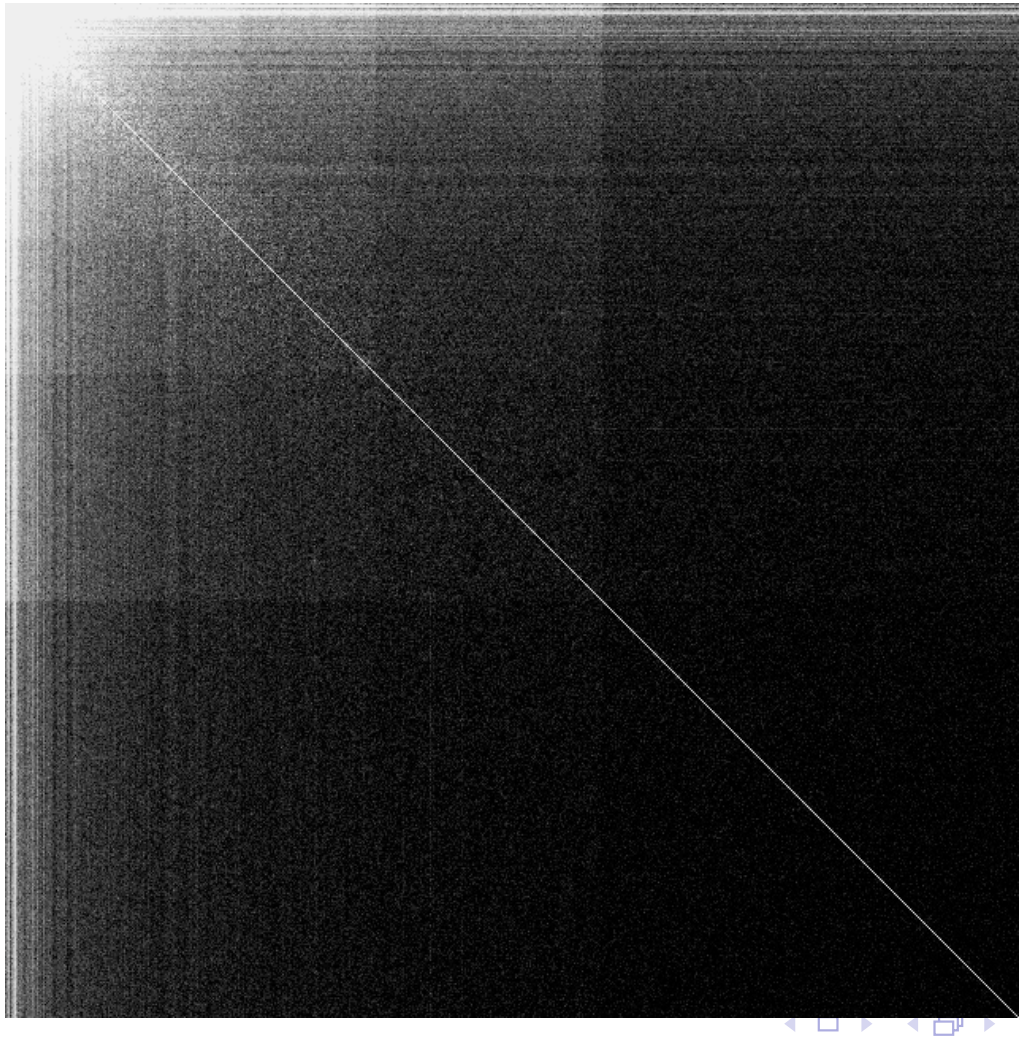
- Old publicly available sponsored search bidding graph from Overture.
- 45k search phrases, 20k advertiser ids, 450k unweighted edges representing bids.

Top phrases in the core decomposition:

	core #	phrase
0	29.0	home loan mississippi
2	29.0	carolina home loan south
4	29.0	mortgage nevada
6	29.0	mortgage texas
9	29.0	alabama home loan
11	29.0	home loan utah
13	29.0	jersey mortgage new
14	29.0	mortgage ohio
17	29.0	home island loan rhode
19	29.0	home kansas loan

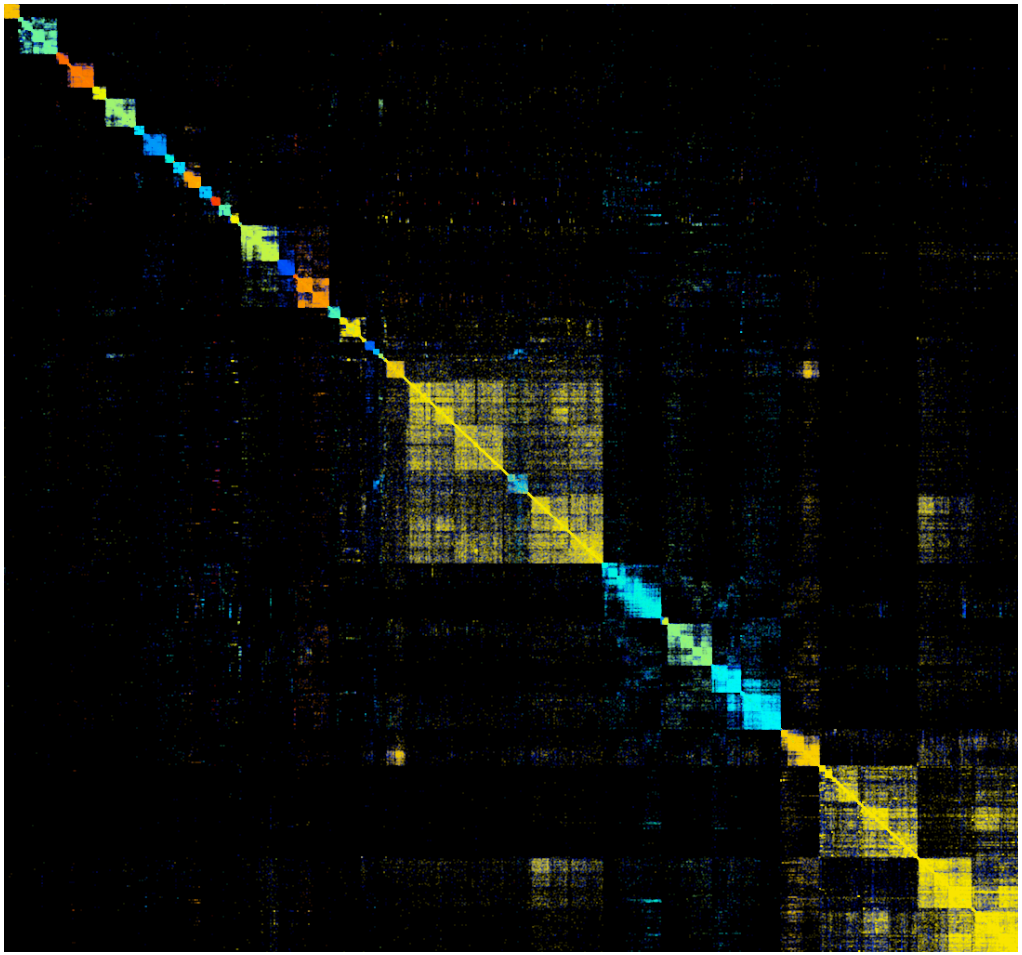
Core ordering in a sponsored search bidding graph

Adjacency matrix, with vertices ordered by the core decomposition



Comparing graph partitioning and core ordering

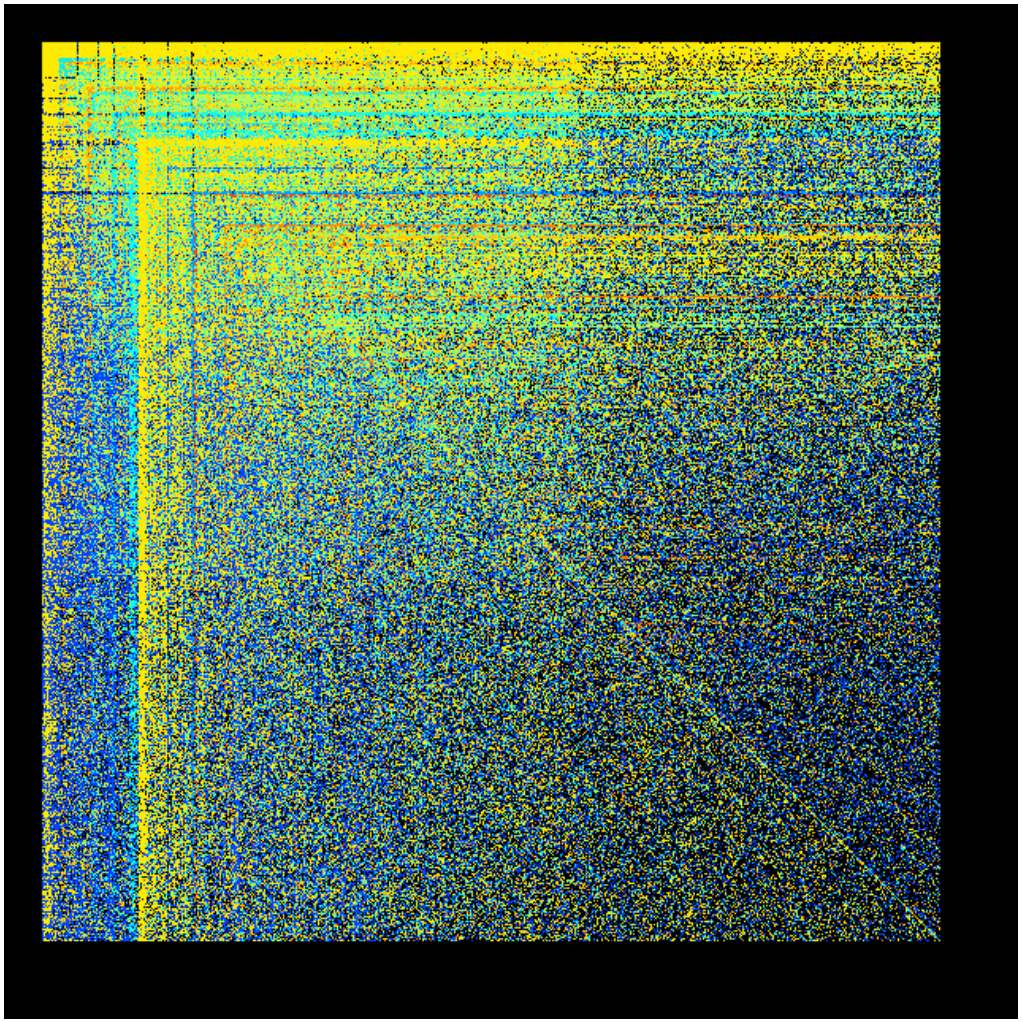
Adjacency matrix of IMDB movie-actress incidence graph, with vertices ordered by recursive graph partitioning, courtesy of Kevin Lang.



The color represents the country in which the movie was

Comparing graph partitioning and core ordering

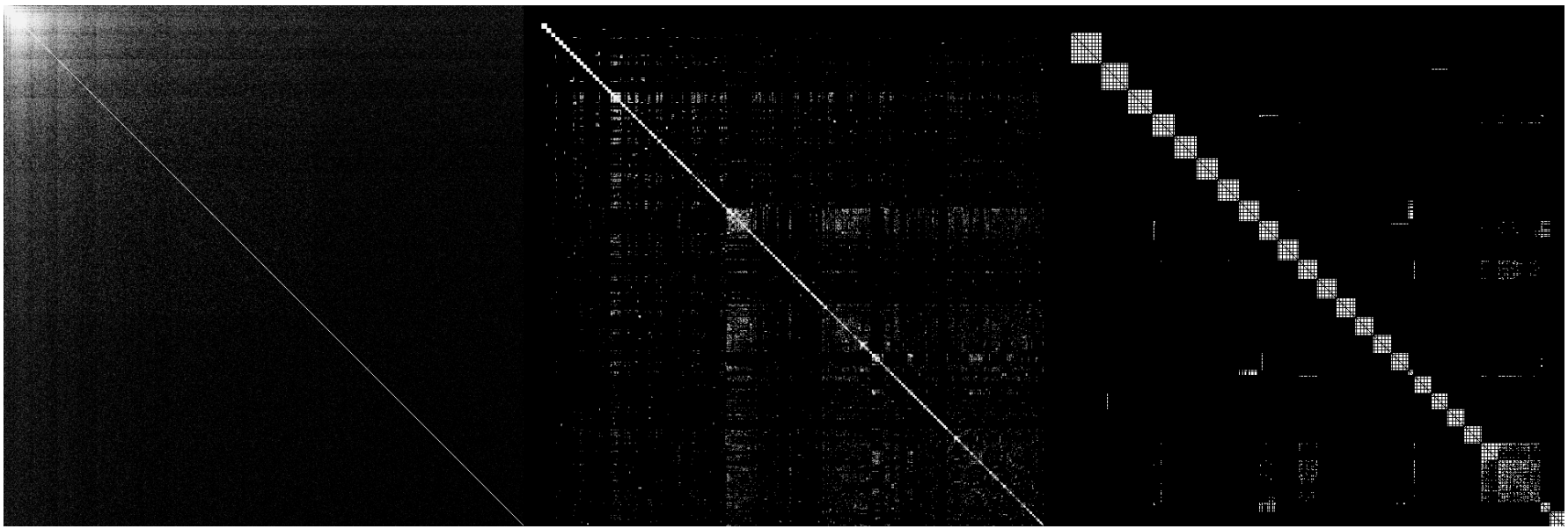
Adjacency matrix of IMDB graph, with vertices ordered by the core decomposition.



The color represents the country in which the movie was

DBLP Collaboration graph

Core ordering as a heuristic for finding cliques.



The end